CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

FEATURE ARTICLE

Jan Axelson

USB Chip Choices

Finding a Peripheral Controller

Today, most peripheral devices are designed with USB connections. Designing USB peripherals can get tricky, but choosing the right chip can make a world of difference. Jan knows her USB, so you might want to choose to listen up.

If you're designing a device that will connect to a PC or Mac, you'll probably use a universal serial bus (USB). You may have noticed that the ports that served PCs for 20 years are disappearing. USB was designed from the ground up to replace a variety of legacy ports with a single interface that's flexible and easy to use.

But simplicity for end users has a price—the interface is more complicated than the single-purpose ports it replaces. To manage the complexity, every USB peripheral must contain an intelligent controller that knows how to respond to the requests defined by the specification. The good news for developers is that there are plenty of choices for controllers.

This article will help you find the USB controller that gives the best performance. I'll start with a quick tour of USB and a review of the responsibilities of USB peripherals. Then, I'll discuss how to narrow the choices. I won't describe every chip, but I will present advantages and disadvantages of some popular chips.

USB, IN BRIEF

USB is suitable for nearly any application that needs a slow to moderate-speed connection to a host CPU with USB support. This article will concentrate on Windows 98 and 2000 hosts, but a host can be any computer with host-controller hardware and operating system support. USB peripherals include standard devices like keyboards, mice, and printers, as well as test instruments, control systems, and other small-volume or custom designs. Video and other high-speed applications will most likely use IEEE-1394/Firewire.

One goal of USB is freeing users from technical and logistical hassles. There's no need to assign IRQs or port addresses. Inexpensive hubs make it easy to add peripherals without having to open the box and find a slot. There's only one interface. And the interface can provide up to 500 mA at a nominal 5 V, so many peripherals no longer need a wall wart or AC power cord for an internal supply.

The host controls the bus and keeps track of which devices are attached. It also ensures each data transfer gets a fair share of the time. Inside the peripheral, the controller hardware and embedded code respond to transmissions from the host.

USB is the product of a consortium that includes Intel, Microsoft, and other companies. The organization, the USB Implementers Forum, sponsors a web site (www.usb.org) that has the specification documents and tools for both developers and end users.

HOST COMMUNICATIONS

Even if you're designing only the peripheral side, it's helpful to know

how the host communicates. Windows uses a layered driver model for USB communications. Each driver layer handles a portion of the communication (see Figure 1).

Applications communicate with device drivers (including class drivers) that communicate with the system's bus drivers, which access the USB hardware. Windows includes bus drivers and some class drivers.

For Windows, a device driver for a USB device must conform to Win32 Driver Model (WDM). A WDM driver, supported by Windows 98 and 2000, is an NT kernel-mode driver with power management and plug-and-play.

A device may have its own driver, or use a generic class driver that handles communications with any hardware that conforms to a class specification. Windows adds class drivers with each release (see Table 1). If your device isn't in a supported class, you must provide a driver.

How does Windows decide which driver to use with a device? Every device stores a series of data structures called descriptors. Every Windows system has a variety of INF files, which are text files that match drivers with class codes or vendor and product IDs stored in the descriptors.

When the files detect an attached device, the host performs an enumeration process that requests the descriptors. All devices must know how to respond to the enumeration requests. The host compares the information in the descriptors with the information stored in the system's INF files and selects the best match. Some products provide their own INF files, others use files provided with Windows.

TRANSFERS

USB 1.1 supports two speeds. Full speed is 12 Mbps. Low speed, which is intended for inexpensive devices and devices that need flexible cables, is 1.5 Mbps. The latest release, version 2.0, supports 480 Mbps, but requires new hardware in the host, peripheral, and any hubs between.

A single peripheral's data transfer rate is less than the bus rate and not always predictable. The bus must also carry addressing, status, control, and

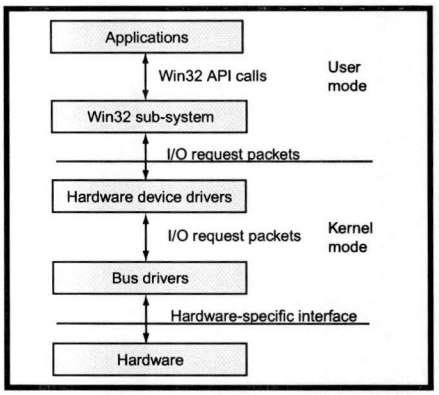

Figure 1—*USB communications use a layered driver model in Windows 98 and 2000. Each layer handles a portion of the communications. Bus drivers and some class/device drivers are provided with Windows.*

error-checking information. Any peripheral may have to share bus time with other peripherals, although a device can request guaranteed delivery rate or maximum latency between transactions. Low-speed transfers are limited to a fraction of the bus time so that they don't clog the bus.

To make the bus practical for devices with different needs, the specification defines four transfer types: control, interrupt, bulk, and isochronous (see Table 2).

Control transfers are the only transfers that every device must support. Enumeration uses control transfers. With each, the host sends a request. The specification defines requests that devices must respond to, and a class or individual device driver may define extra requests.

Along with each control request, the host sends a 2-byte value and a 2-byte index, which the request can define in any way. Depending on the request, either the host or device may send data. The receiver returns an acknowledgement. However, there is no data stage with some requests, and the device returns an acknowledgement after receiving the request.

The other transfers don't use defined requests. They transfer blocks of data and identify and error-check information to or from a device.

Interrupt transfers are useful for applications that need to send small amounts of data at intervals, such as keyboards, pointing devices, and other monitoring and control circuits. A transfer can send blocks of up to 64 bytes with a guaranteed latency (maximum time between transactions) of 1 to 255 ms.

Bulk transfers are useful for applications that need to transfer large amounts of data when delivery time isn't critical, such as printing and scanning. A bulk transfer can send blocks up to 64 KB, but without guaranteed delivery time.

Isochronous transfers are used when delivery rate is critical and errors can be tolerated, such as audio to be played in real time. An isochronous transfer can send up to 1023 Bpms with a guaranteed attempt to send a block of data every millisecond. Unlike the other transfers, isochronous transfers have no handshake packet that enables the receiver to notify the sender of errors detected within data that is received.

USB transfers consist of one or more transactions. Each transaction, in turn, contains identifying information, data, and error-checking bits.

Inside the device, all USB data travels to or from an endpoint, which is a buffer that stores data to be sent or received. A single device can have up to 16 endpoint numbers (0–15). An endpoint address is the endpoint number plus its direction: in (device-to-host) or out (host-to-device). Every device must support endpoint 0 in and out for control transfers and may support up to 30 additional endpoints.

Most controllers support fewer than the maximum number of endpoints and some don't support all of the transfer types. Low-speed controllers are limited to using control and interrupt transfers. Cypress Semiconductor's EZ-USB is one chip that supports the maximum number of endpoints (one bidirectional control endpoint plus 30 additional endpoints) and all four transfer types.

The host controls the bus and initiates transfers. But, a device in the low-power suspend state can use the remote wake-up feature to request a transfer. And a device can request the host to send or request periodic interrupt or isochronous data.

ELEMENTS OF A USB CONTROLLER

A USB peripheral controller has several responsibilities. It must pro-

Windows edition	USB device drivers added
Windows 98 Gold (original)	audio HID 1.0 (includes keyboard and pointing devices)
Windows 98 SE (second edition)	HID 1.1 communications (modem) still image capture (scanner, camera), (first phase/preliminary)
Windows 2000 Windows 98 Millennium	mass storage printer

Table 1—*Each release of Windows added drivers for new classes of USB devices. If your device fits into one of the supported classes, you don't need to write a driver for it.*

vide a physical interface to the bus and detect and respond to requests and other events at the USB port. And it provides a way for an internal or external CPU to store data that it wants to send and retrieve.

Controller chips vary by how much firmware support they require for these operations. Some, such as NetChip's NET2888, require little more than accessing a series of registers to configure the chip and store and retrieve bus data. Others, such as Cypress' M8 series, require routines to manage data transfers and ensure that the appropriate handshaking information is exchanged.

Some chips use registers, and others reserve a portion of data memory for transmit and receive buffers.

For faster transfers, Philips Semiconductor's PDIUSBD12 has double buffers that store two full sets of data in each direction. While one block of data is transmitting, the firmware can write the next block of data into the other buffer so it's ready when the first block finishes transmitting. In the receive direction, the extra buffer enables a new transaction's data to arrive before the firmware finishes processing data received in the previous transaction. In both cases, the hardware automatically switches between the buffers.

A controller likely will have an interface other than the USB port to the outside world. In addition to general-purpose I/O pins, a chip may support other serial interfaces, such as an asynchronous interface for RS-232 or synchronous interfaces, such as I^2C or Microwire.

Some chips include special interfaces. For example, Philips' USA1321 contains a digital-to-analog converter (DAC) for USB speakers and other audio devices. NetChip's NET1031 is a scanner controller with a USB interface. Dallas Semiconductor's DS2490 is a USB-to-1–wire bridge.

SIMPLIFYING THE PROCESS

Aside from the chip's features, easy development affects how long it takes to get a project running. The simplest and quickest USB project meets the following criteria. First, you must be familiar with the project's chip architecture and programming language. Second, the project has a development system that enables easy firmware downloading and debugging. Third, it has detailed, well-organized hardware documentation. Fourth, there is well-documented, bug-free sample firmware for an application similar to your project. And fifth, it can communicate using device drivers included with Windows or another well-documented driver that you can use with minimal modification.

These are not trivial considerations. The correct choice will save many hours and much aggravation.

ARCHITECTURE CHOICES

Some USB controllers contain a general-purpose CPU, and others have a serial or parallel interface that must connect to an external CPU.

A chip with a general-purpose CPU may be based on an existing family such as the 8051, or may be designed specifically for USB applications. Controllers that interface with an external CPU provide a way to add USB to any microcontroller with a data bus. The external CPU manages non-USB tasks and communicates with the USB controller as needed.

For applications that require fast performance, another option is to design an application-specific integrated circuit (ASIC). Components are available as synthesizing VHDL and Verilog source code.

Cypress has several chips that contain a CPU developed specifically for USB applications. The M8 family includes the CY7C6*xxx* series of inexpensive chips, each with two to four endpoints, 12 to 32 general-purpose I/O lines, and 2 to 8 KB of program memory. Note that the program memory is one-time programmable (OTP) EPROM.

The instruction set is short (35 instructions), so learning it isn't difficult. However, this also means you won't find detailed instructions that do most of the work for you. For example, there are no instructions for multiplying or dividing; calculations must be done by adding, subtracting, and bit shifting (Cypress offers a C compiler from Byte Craft with extensive math functions).

For 8051 users, Cypress' EZ-USB has an architecture similar to Dallas Semiconductor's 80C320. Two other early 8051 compatibles were Intel's 8*x*930 and 8*x*931. Intel stopped manufacturing both of these this year but licensed the technology to Cypress.

If you have 8051 experience, especially if you're designing a USB-capable version of an existing 8051 product, sticking with the 8051 makes sense. Even if you're not familiar with the architecture, its popularity means that programming and debugging tools are available, and you're likely to find sample code and advice from other users on the Internet. Keil has C compilers for the

Transfer type	Required	Low speed OK	Error correction	Guaranteed delivery rate	Guaranteed maximum latency	Typical use
control	Y	Y	Y	N	N	enumeration
bulk	N	N	Y	N	N	printer
interrupt	N	Y	Y	N	Y	mouse
isochronous	N	N	N	Y	Y	audio

Table 2—*The USB's four transfer types are designed to meet different application needs.*

8x930/1, and both Keil and Tasking have a C compiler for the EZ-USB.

Other examples of families with USB-capable chips are Mitsubishi's 740, 7600, and M16C, Motorola's HC05, and Microchip's PIC16C7x5.

Controllers that interface to external CPUs typically use a parallel or synchronous serial interface. An interrupt pin signals the CPU when the controller receives USB data or is ready for new data to send. This works if you want to use a CPU that doesn't have a USB-capable variant.

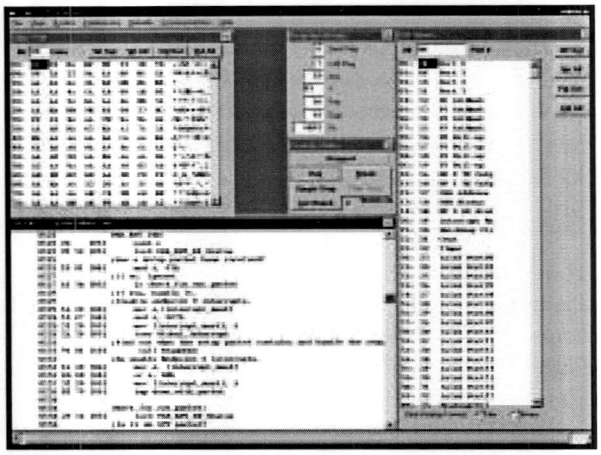

Photo 1—*Cypress Semiconductor's M8 Monitor program enables you to control program execution, and view and change memory and registers.*

Philips' PDIUSBD11 has an I²C interface that uses three pins, a clock input, bidirectional data, and an interrupt output. The maximum clock frequency of the chip's I²C bus is 100 kHz, so it doesn't handle high-volume transfers. In contrast, the PDIUSBD12 has a multiplexed parallel bus that can transfer up to 1 Mbps.

National Semiconductor's USBN9602 can be configured to transfer multiplexed or non-multiplexed parallel data or Microwire serial data.

DRIVER CHOICES

The other side of programming a USB device is the device driver and application software on the host. You can use a device driver included with Windows, use or adapt a driver from another source, or write your own.

The human interface device, known as HID, drivers included with Windows 98 and 2000 are an option for general-purpose applications up to 64 KBps. HIDs can use control and interrupt transfers.

The classic HID examples are the keyboard and mouse, when a human's actions cause data to be sent to the host. But, a HID doesn't need a human interface, it can include test instruments, control circuits, and other devices that operate within the limits of the class specification.

Applications access HIDs using the API functions `ReadFile` and `WriteFile`. The device's firmware must include the HID class code in its descriptors and define a report format for the data it will exchange. The report format tells the host the size and quantity of the report data, and also may provide units and other information to help the host interpret the data.

The mass-storage driver introduced with Windows 2000 is an option for devices that need to transfer a lot of data but don't have critical timing requirements.

For custom drivers that use bulk or isochronous transfers, start with the `bulkusb.sys` and `isousb.sys` examples in the Windows 2000 DDK. If you use these, search the Developers Webboard at www.usb.org for tips and bug fixes.

DEBUGGING TOOLS

Most chip vendors offer a development board and basic debugging software to make development easier. The development board enables you to load a program from a PC to the chip's program memory, or to circuits that emulate the chip's hardware.

Typical debugging software uses a monitor program, which enables you to control program execution and watch the results (see Photo 1). You can step through a program line by line, set breakpoints, and view the contents of the chip's registers and memory. And, you can run the monitor program and a test application at the same time. Look inside the emulated chip to view registers and memory contents as your application communicates with it.

Another useful debugging tool is a USB protocol analyzer. Because the data on the bus is encoded, conventional oscilloscopes and logic analyzers aren't helpful for viewing USB data. A protocol analyzer captures the data, then filters and displays it in a variety of formats. PC-based analyzers may connect to an Ethernet port or an ISA card. Other analyzers are designed as attachments to logic analyzers.

PROJECT NEEDS

In addition to looking for a chip that will be easy to work with, narrow the choices by specifying your project's requirements and looking for chips that can meet them. Here are some questions to consider.

How fast does the data need to transfer? The rate of data transfer depends on several things: whether the device is low- or full-speed, how busy the bus is, and the transfer type. As a peripheral designer, you don't control how busy a user's bus will be, but you can design your product to work in a worst-case scenario.

If a product requires only occasional interrupt and control transfers, a low-speed chip may save money. But, the fastest configuration for a low-speed interrupt endpoint is 8 bytes per transaction with a maximum latency of 10 ms between transactions, which is 800 Bps.

How many and what type of endpoints do you need? Each endpoint is configured to support a transfer and direction. Although the host can request a new configuration or interface to use a different transfer for each, in most cases each transfer type and direction will have its own endpoint.

What about firmware upgrades? For program memory, many USB devices use EPROM, in which changing the firmware requires removing the chip. The EZ-USB supports an easier way, using a re-enumeration process that loads the program code into the chip from the host on each power-up. If you expect firmware changes, the EZ-USB is difficult to beat.

Do you need a flexible cable? One reason why most mice are low-speed devices is that the less stringent re-

quirements for a low-speed cable mean that the cable can be thinner and more flexible.

Need a long cable? Low-speed cables are limited to 3 meters, and full-speed cables can be 5 meters. Full-speed cables have shielded, twisted pairs. Hubs can stretch a connection beyond these limits. The limit is five hubs plus the host, each with a 5-meter cable, for a maximum distance of 30 meters. Active extension cables that contain embedded hubs are available.

What other hardware features and abilities are needed? The list includes everything from general-purpose I/O to on-chip timers. The requirements depend on the application.

The answers to these should narrow your search, making your chip choices and the development as painless as possible.

This article is adapted from material in *USB Complete: Everything You Need to Develop Custom USB Peripherals* by Jan Axelson.

Jan Axelson has worked with electronics and computers for 25 years. Jan's web site (www.lvr.com) has resources for developers using USB and legacy ports. You may reach her at jan@lvr.com.

REFERENCES

USB Central, links for USB developers, www.lvr.com/usb.htm.

USB Designer Links, links to USB controller chips, www.ibhdoran.com/usb_link.html.

USB Implementers Forum, the specification documents, Developer's Webboard, and more, www.usb.org.

SOURCES

USB Chips

Cypress Semiconductor
(408) 943-2600
Fax: (408) 943-6848
www.cypress.com

Dallas Semiconductor
(972) 371-4448
Fax: (972) 371-3715
www.dalsemi.com

Keil Software
(800) 348-8051
(972) 312-1107
Fax: (972) 312-1159
www.keil.com

Microchip Technology, Inc.
(888) 628-6247
(480) 786-7200
Fax: (480) 899-9210
www.microchip.com

Mitsubishi Electronics
(408) 730-5900
Fax: (408) 730-4972
www.mitsubishichips.com

Motorola
(512) 328-2268
Fax: (512) 891-4465
www.mot-sps.com/sps/general/chips-nav.html

National Semiconductor
(408) 721-5000
Fax: (408) 739-9803
www.national.com

NetChip Technology, Inc.
(650) 526-1490
Fax: (650) 526-1494
www.netchip.com

Philips Semiconductor
(408) 991-5207
Fax: (408) 991-3773
www.semiconductors.philips.com

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, subscribe@circuitcellar.com or www.circuitcellar.com/subscribe.htm.